Name: **Sourojit Saha**
Andrew id: **sourojis**
Course: **16-782 Planning and Decision Making in Robotics**
Assignment: **Assignment 2**

1. Algorithms

RRT Planner

The node structure used contains the following information of a node:

- Configuration of itself.
- Its id. This is used to uniquely identify a node.
- The id of its parent.

The nodes are stored in a vector. The tree is initialised by inserting the start position as a node to the vector. Then, until the goal is found, a random sample was generated. To speed up the planner, a goal biasing was also added. This enabled that the goal is sampled with a probability of 50%.
After the sample has been generated, the nearest node in the tree is searched and a unit vector from this node to the sample is computed. After this, the nearest node from the tree is extended by ϵ amount along the unit vector. After trial and error, the value of epsilon has been chosen as 0.4 for optimal results. The validity of the configuration was checked during extension. If the extension produced collisions, it was discarded and new sample was generated. If the extended node was found to be in the neighborhood of the goal, it was considered as reaching the goal and the next configuration sampled was the goal configuration. The neighborhood radius has also been kept as ϵ, so that the distance between two successive nodes in a tree is never greater than ϵ.
To speedup the planner, a chunk of memory was reserved for the tree at the initialization.

RRT Connect Planner

The node structure used contains the following information of a node:

- Configuration of itself.
- Its id. This is used to uniquely identify a node.
- The id of its parent.
- A flag that stores which tree the node belongs to.

The nodes were stored in a vector. Two trees were initialized. One starting from the start node/configuration and the other starting from the goal node/configuration. The first node in the start tree is the start node and the first node in the goal tree is the goal node. Then, until the solution was found, the sample was generated at random. Starting with the start tree, a sample is generated and its closest neighbor in the start tree is searched. The unit vector from this node to the sample is computed and the node is extended in that direction by an ϵ amount. The value of ϵ was set to 0.4. This value was observed to give the optimal result after trying multiple values. If the extended configuration was invalid, the sample was discarded and another sample was generated. After the extending the node, the nearest node from the other tree was searched. The unit vector from this node to the extended node was computed and extended by ϵ till it reached the node or hit an obstacle. Now the trees were swapped and the process repeated, till the stopping criteria was not fulfilled. When the nearest nodes

of the two trees were within ϵ from each other, it was considered as finding the solution and planner was stopped. The path was then generated by backtracking both the trees from their respective nearest nodes. To speedup the planner, a chunk of memory was reserved for the tree at the initialization.

RRT Star Planner

The node structure used contains the following information of a node:

- Configuration of itself.
- Its id. This is used to uniquely identify a node.
- The id of its parent.
- Cost to reach that node from start.

The nodes were stored in a vector. The tree was initialised by inserting the start configuration as a node to vector. Till the goal was found, a sample was generated at random. To speed up the planner goal biasing was added. This enabled that the goal was sampled with a probability of 50%.
Similar to RRT, the nearest node was searched and was extended in the direction of unit vector from this node to the sample by an amount of ϵ. The value of ϵ was set at 0.4, after trying with multiple values, for optimal results. Then a re-wiring step was done where the nodes within a certain radius were searched. Then the path cost from all these nodes to the extended node was calculated and an edge was added between the extended node and the node with least cost to extended node. After this step, the second re-wiring step was executed where the cost to the nodes in the neighborhood through the extended node was calculated and the path providing the least cost was updated. If the extended node was within ϵ neighborhood of the goal, it was considered as reaching the goal and the planner was stopped. The path was generated by backtracking the nodes from the goal to start. To speedup the planner, a chunk of memory was reserved for the tree at the initialization. The radius for searching in neighborhood was calculated as:

$$radius = min(\frac{\gamma}{\delta} \cdot (\frac{log|V|}{|V|})^{1/d}, \epsilon) \tag{1}$$

Where $|V|$ is the number of vertices, d is the dimensionality, δ is the volume of unit hypersphere, and ϵ & γ are user defined parameters. After trial and error with multiple values, the value of $\frac{\gamma}{\delta}$ was set as 100.

PRM Planner

The node structure used contains the following information of a node:

- Configuration of itself.
- Its id. This is used to uniquely identify a node.
- The id of its parent.
- List of neighbors.
- Cost to reach that node from start.
- Whether the node has a link connecting it to start node.
- Whether the node has a link connecting it to goal node.

The nodes were stored in a vector. The graph was initialised by inserting the start and goal configurations as nodes. Till the stopping criteria was reached, a random sample was generated. Its validity was checked, if it was found to be invalid, it was discarded and another sample was generated. Then nearest neighbors within a distance of 2ϵ were searched. These nodes were added as neighbors of this sample node. The collision checking was performed for all the edges. Then for all the closest neighbors, the optimal path through that node to the sample node was calculated and set as the cost of the sample node. After this, it was checked whether the sample node has a link connecting it to start and another connecting it to goal. If this condition was fulfilled, it was considered as reaching the goal. The planner was stopped and a Dijkstra search was initiated to generate the path from start to goal.

Note: All the distances and metrics used were Euclidean distance.

2. Results

The following configurations were generate randomly and used for testing:

Tabel 1: Testing Samples generated randomly

Problem Index	Start Configuration	Goal Configuration
0	1.8326,0.785398,4.95674,1.51844,0.977384	1.13446,2.14675,2.25147,2.86234,1.93732
1	1.51844,1.29154,4.95674,2.14675,4.01426	0.10472,1.62316,2.49582,1.98968,4.43314
2	0.366519,1.72788,2.37365,0.20944,4.10152	1.44862,5.84685,2.26893,1.71042,1.5708
3	0.314159,2.72271,0.837758,1.27409,2.23402	1.32645,1.32645,3.83972,2.30383,0.785398
4	1.54566,0.890118,2.0944,3.33358,1.27409	0.837758,3.47321,0.0872665,1.5708,5.16617

The results for the above configurations as generated by *grader.py* is given below. Note that the column for *Nodes Generated* has been added later as asked in the report.

Tabel 2: Results Generated by *grader.py*

Planner	Map	Problem Index	No. Steps	Cost	Time	Nodes Generated	Sucess
0	./map2.txt	0	21	7.587	0.003	23	true
0	./map2.txt	1	16	4.994	0.004	20	true
0	./map2.txt	2	75	27.571	0.505	1665	true
0	./map2.txt	3	26	9.908	0.007	73	true
0	./map2.txt	4	36	14.069	0.021	303	true
1	./map2.txt	0	19	7.089	0.004	19	true
1	./map2.txt	1	17	5.220	0.004	17	true
1	./map2.txt	2	107	39.791	0.044	546	true
1	./map2.txt	3	28	10.062	0.004	43	true
1	./map2.txt	4	41	15.477	0.007	147	true
2	./map2.txt	0	11	7.853	0.004	11	true
2	./map2.txt	1	8	4.782	0.004	10	true
2	./map2.txt	2	22	15.785	0.028	308	true
2	./map2.txt	3	14	10.051	0.004	25	true
2	./map2.txt	4	18	13.549	0.167	847	true
3	./map2.txt	0	13	11.222	13.631	19296	true
3	./map2.txt	1	11	9.529	3.757	12077	true
3	./map2.txt	2	16	14.015	53.652	32277	true
3	./map2.txt	3	14	12.148	13.756	19477	true
3	./map2.txt	4	16	14.199	12.545	191930	true

Tabel 3: Overall performance of the Planners

Planner	Avg Time	Avg Cost	Avg Steps	Avg Nodes
RRT	0.108	12.826	34.800	416.800
RRT Connect	0.013	15.528	42.400	154.400
RRT Star	0.041	10.404	14.600	240.200
PRM	19.468	12.223	14.000	55011.400

Tabel 4: Sucess Rate

Planner	No. of Soln. under 5s
RRT	5
RRT Connect	5
RRT Star	5
PRM	0

We see that all the planners execute under 5s, except PRM planner. This is because PRM uniformly samples the whole region and grows the graph in all directions, consuming more compute and time.

3. Comparison

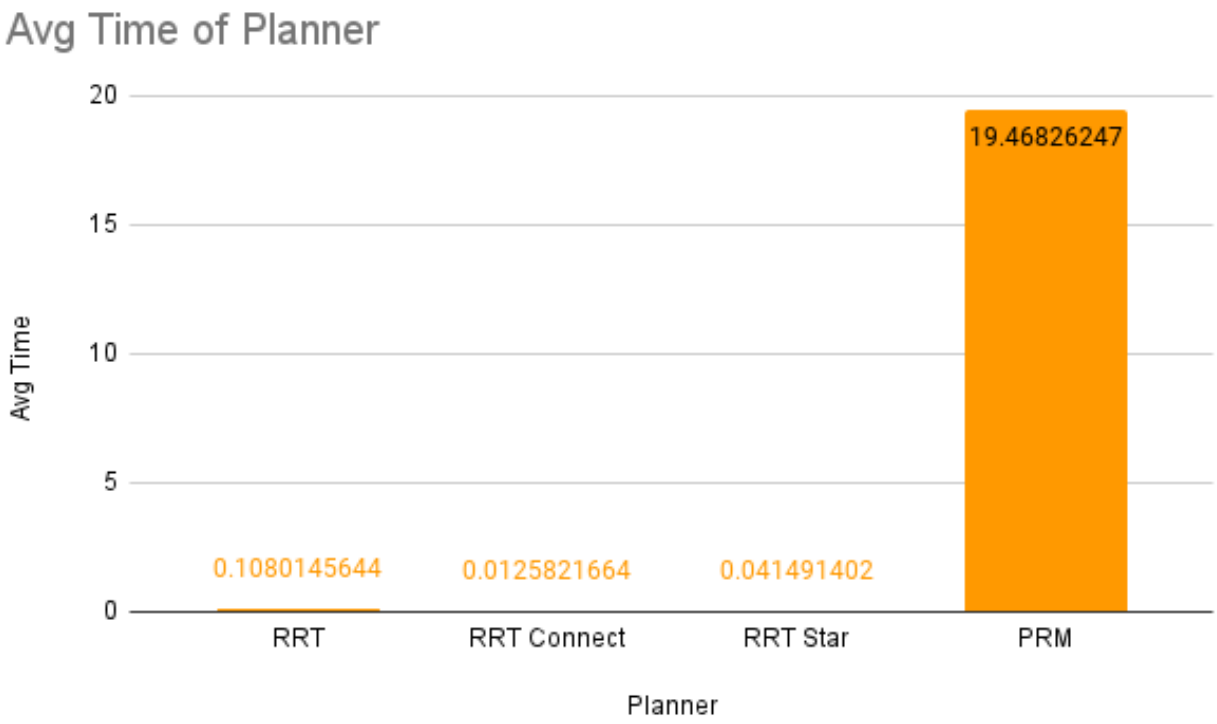

Figure 1: Plot for Avg Time of Planner

RRT Connect performs the best w.r.t time. This is reasonable as it keeps on extending the tree until it finds the solution. RRT Star takes more time than RRT Connect, this is because the rewiring step in computationally expensive. RRT performs better than PRM as it has a goal biasing implemented that allows the tree to grow towards the direction of the goal. PRM on the other hand takes the most time as there is no goal biasing implemented in the current implementation. This causes it to explore the whole region uniformly without any bias towards reaching the goal.

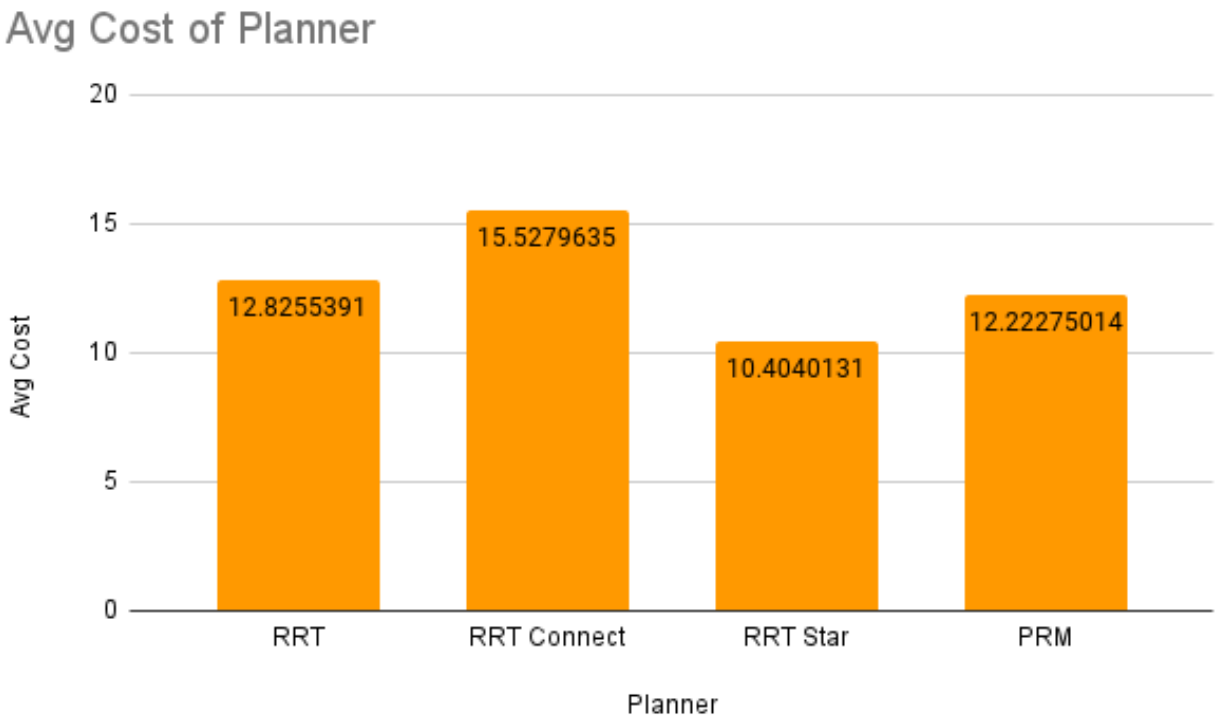

Figure 2: Plot for Avg Cost of Planner

RRT Star performs the best w.rt. cost. This is reasonable as it optimizes over cost by the two re-wiring steps. The cost for RRT is greater than RRT Star, this is expected as RRT Star is a modification over RRT where it optimizes over cost. The cost of PRM is comparable to RRT. This may be because of the reason that PRM searches uniformly over the full configuration space. RRT Connect performs worst w.r.t cost. This is reasonable as it does not optimize over cost, instead it optimizes over time. This is evident from the above plot.

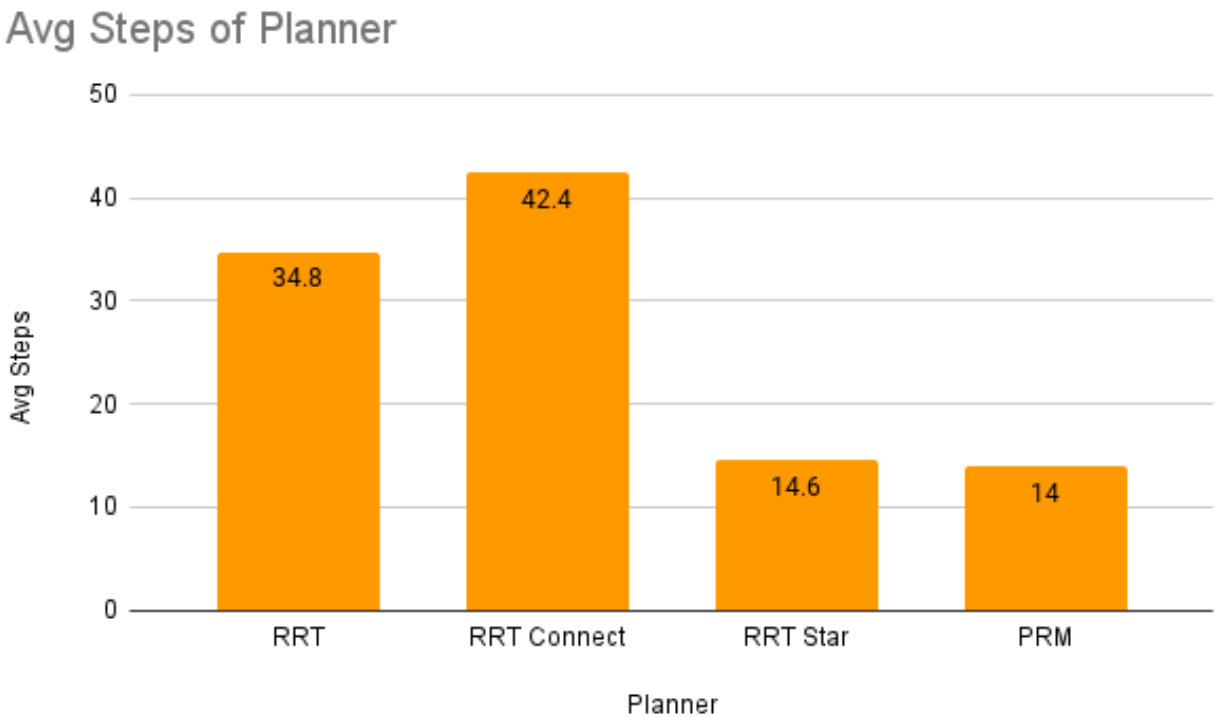

Figure 3: Plot for Avg Steps of Planner

The RRT Star and PRM, both have similar performance w.r.t steps They also perform better than RRT and RRT Connect. This is reasonable as both RRT Star and PRM use the information about the cost to make edge connections. The cost and steps are correlated as an increase in step will most likely also increase the cost. This correlation is verified from the Plot for Avg Cost of Planner, where RRT Star and PRM perform better than RRT and RRT Connect. And RRT Connect performs worst in both cases.

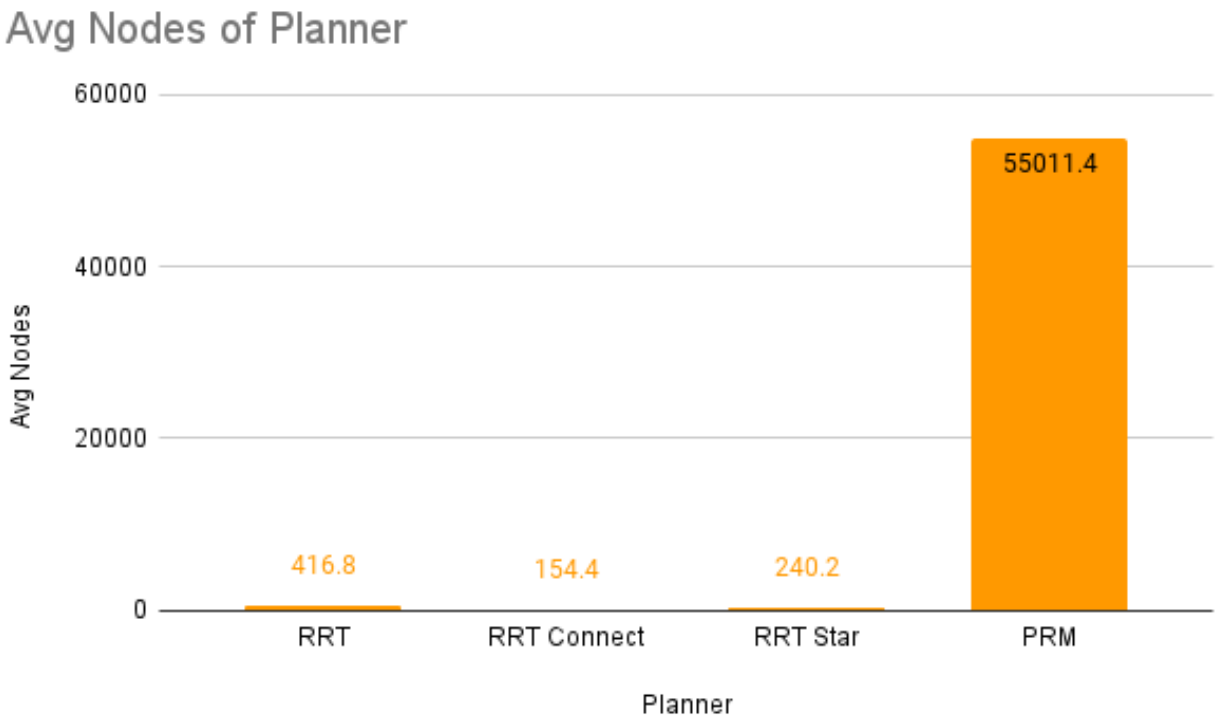

Figure 4: Plot for Avg Nodes of Planner

As Expected, RRT Connect explores the least number of nodes. This is because it prioritises the connecting of two trees and keeps on extending a single branch of the tree till it connects to the other tree or hits an obstacle. RRT Connect is followed by RRT Star and RRT. Though both have goal biasing implemented, RRT Star explores less number of nodes, this is because it rewires itself in every iteration based on optimal cost to get the optimal path. RRT on the other hand does not have any rewiring step, which makes it less optimal than RRT Star. PRM explores the most number of nodes. This is because it explores the whole region uniformly and grows the graph in all directions uniformly.

Based on the above analysis:
1. Assuming the given manipulator will be most likely be setup as an industrial manipulator in an industrial production line and speed is more important than cost, I will prefer RRT Connect planner. This is because this planner is fast and is suitable for planning in real time, and can replan in case of a detection of dynamic obstacle(such as humans). This will aid in a faster production rate.
2. Some of the configurations are not optimal and give rise to high cost. In this application, it can be high motor torque, which will require more energy to execute. It is desirable to have a planner that also reduces the cost.
3. A combination of RRT Star and RRT Connect will be ideal in my opinion. The RRT Star will be the main planner running, as it gives the path of least cost among all, and has a reasonable planning time. Whenever, there is a disturbance in the environment and the environment changes, the planner can fall back to RRT Connect, which will quickly bring it to a stable configuration, and then RRT Star can again take over.

4. Extra Credits

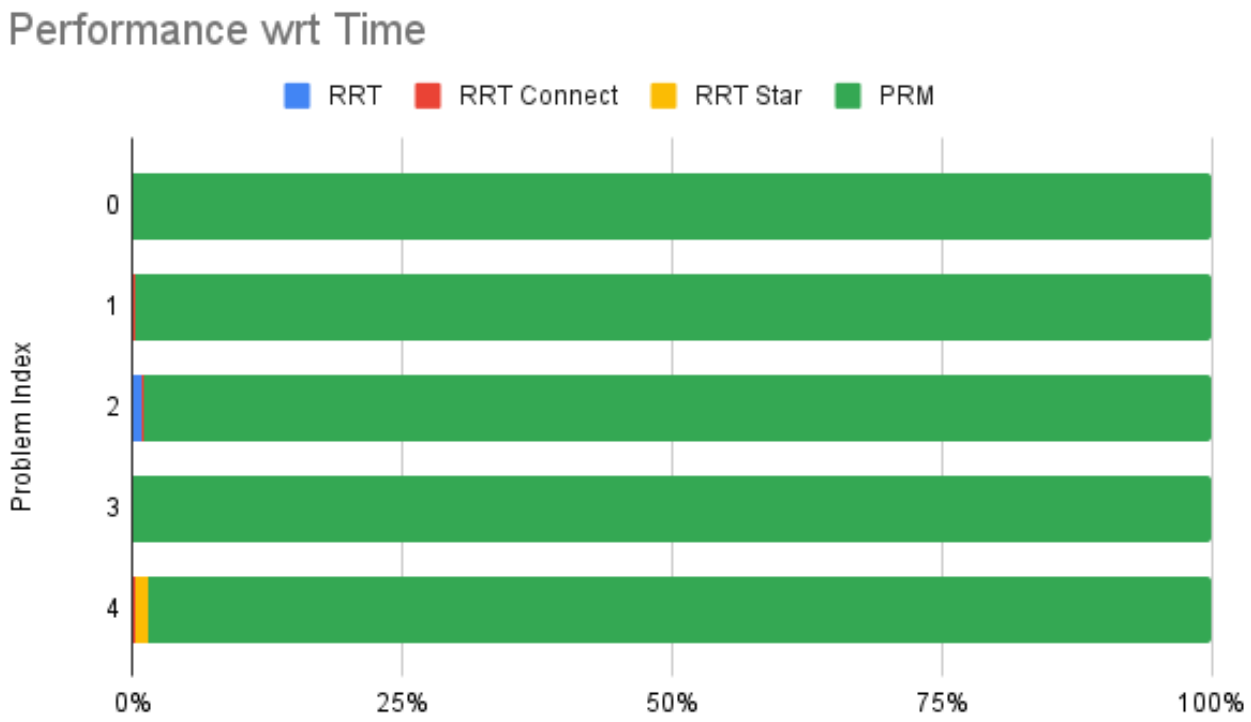

Figure 5: Plot for Performance wrt Time

As can be seen PRM consistently performs worst for all problem. This is because it uniformly searches the configuration space, utilizing more time.

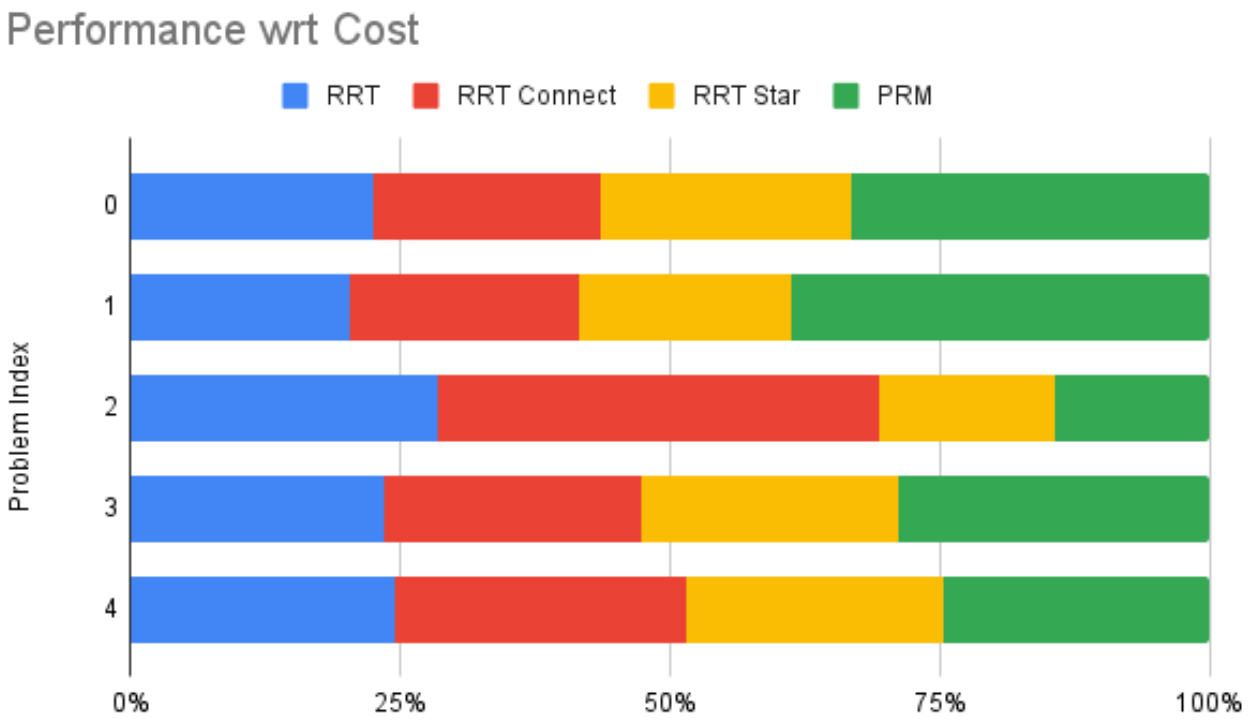

Figure 6: Plot for Performance wrt Cost

RRT Star always performs better than other planning algorithms.

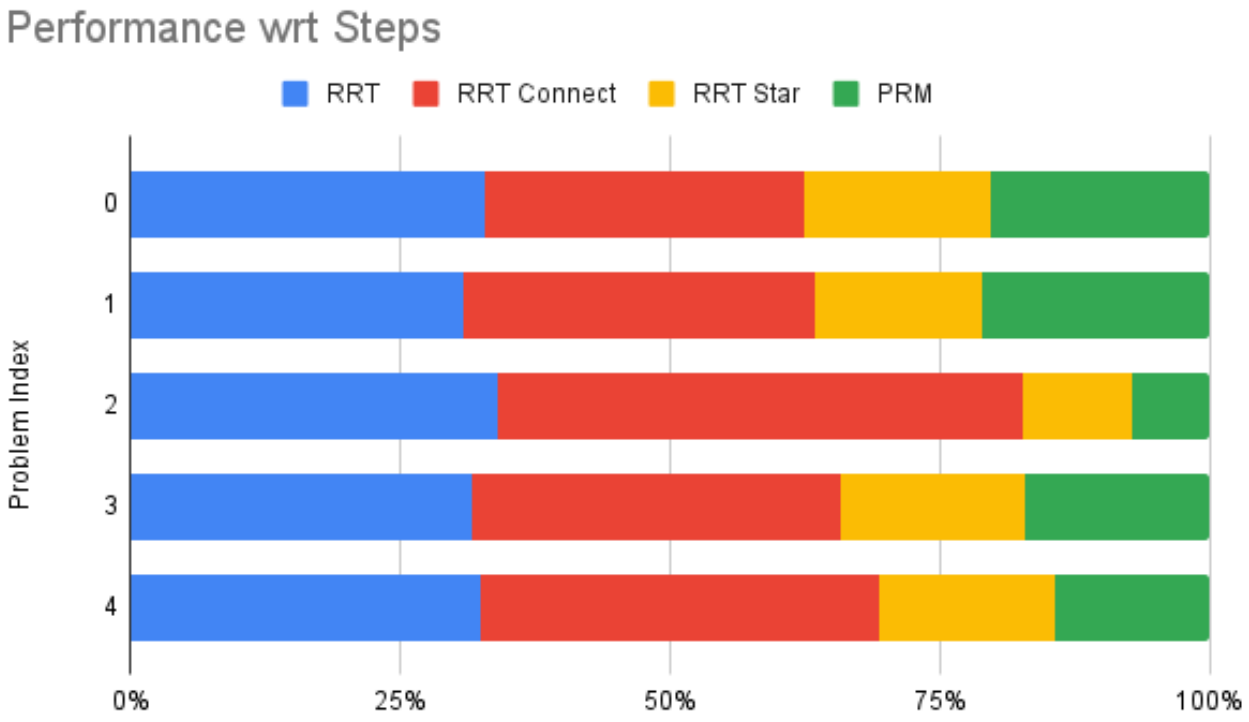

Figure 7: Plot for Performance wrt Steps

RRT connect always takes more steps. This is reasonable as it prioritizes connecting trees.

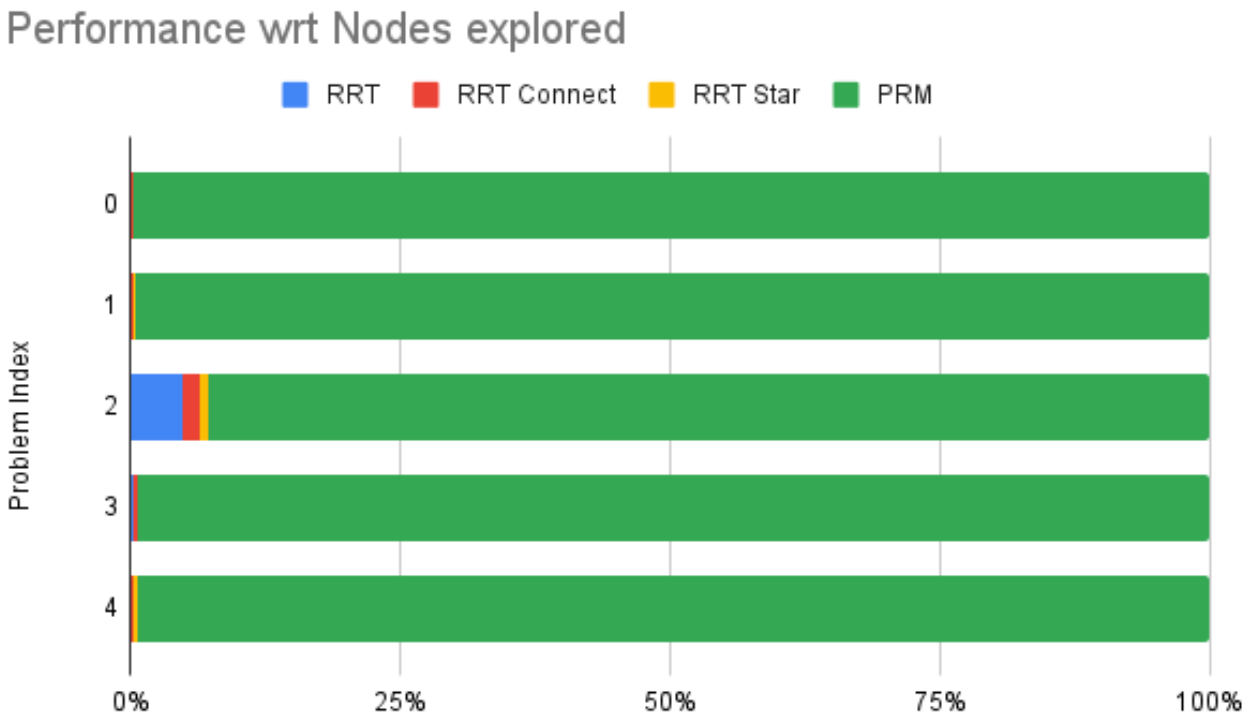

Figure 8: Plot for Performance wrt Nodes Explored

PRM always explores more nodes. This is because it uniformly samples the configuration space and is not directed specifically towards the goal.

Effect of ϵ:
The value of epsilon shows the step size in the hyper plane that the robot/planner can take. A smaller value of ϵ would mean a slower step size. This will result in a more finer and smooth trajectory, but will consume a lot of time because the rate of exploration has now reduced. If the value of ϵ is kept higher, this will result in a faster trajectory generation and trajectory will not be smooth. The value of ϵ has to be set after studying the tradeoff between time to plan and the smoothness of the plan.

5. Compilation Instructions

Compile as follows:

- g++ planner.cpp -o planner.out
- ./planner.out [mapFile] [numofDOFs] [startAnglesCommaSeparated] [goalAnglesCommaSeparated] [whichPlanner] [outputFile]
  Example: ./planner.out map1.txt 5 1.57,0.78,1.57,0.78,1.57 0.392,2.35,3.14,2.82,4.71 2 myOutput.txt